胡文 Go.ogle

作者: Fango

初稿: 2010年11月11日

修订: 2011年10月10日

目录

第一章

临窗望海，空气使玻璃不可能再干净。哪里还有海鸟，没有了鱼的海湾，是谁用垃圾填满。海鸟们飞去了哪里？可否找到一个栖息地，有干净的水梳洗羽毛，有干净的鱼喂可爱的雏，有干净的天愉快的飞。

只好缩在我的玻璃壳里了。可怜的心肺都已过劳。庆幸过劳的日子能换回个壳，不会流浪老狗般垂涎街尾。尽管壳是玻璃的，我只能希望不要震碎，也没人丢石子。

回到电脑前，除了气管食道，和久久未用的阴道，我决定这是我最后也最终的输入端口了。早已没心情看时事如绯闻般胡闹，也早已麻木到不看有聊的淫民厌恶着无趣的蚁们追逐炫耀。我的心是止的。思是脉动的。我要电脑的插入。要电脑的时钟。要电脑的语言。

要他的世界。

我受够了电脑。拿着计算机科学的本，打着信息技术的工，吞着时髦架构的枣，吐的全是糟柏。糟柏。看别人陶醉在自酿的酒中，我在想

Something is wrong, but not me.

我想要明白电脑的世界。要从他的语言开始。电脑的语言是给人说的。是说给别人听的。但要用一套电脑能自动翻译的方式说。所以电脑编程语言是人语，是一套使用特殊标点符号的，简单到连电脑都能懂的英语方言。可为什么我已经会说这么多国的电脑英语方言了，还是不懂电脑在干什么和要他怎么做呢？在电脑面前，是我有诵读障碍，还是我缺乏一套直接的，明白无误的，有超强表达能力的，那么一种方言呢？

2009年11月10日

照例打开Ars technica (拉丁语，技术的艺术的意思）。新出的Go: 谷歌的电脑方言。没什么大不了。每个电脑公司都制造自己的方言来圈养一群忠实的牛羊，繁殖在自己的框架里。升阳的 Java 和 J2EE，微软的 C# 和 .Net，苹果的 Objective C 和 cocoa，每每让我张口结舌口吃不已。

我对谷歌还是有多些好感的。“You can make money without doing evil”。说她想有道德也想赚钱。问题是不管是爱财有道还是想表牌坊，多一套方言终归是不好的，她想以德服务人母泽天下，但群’雄‘难伺啊。

2010年5月20日

“Google IO”。庵卓和网舍开发者的盛宴，Java和Javascript程序员的狂欢。和我无关的被我关心着。

但有两个名字被注意了我。Rob Pike和Russ Cox。技术大拿们。我手里的三键无滚轮鼠标就是心甘情愿被你们强加的。你们玩的Plan9 Inferno让我第一次完全看清了操作系统的虚幻本质。你们的ACME成了我唯一首选的编辑器。你们是Go的作者吗？

google “google go” 直接带我到了他的网舍 http://golang.org。朴素而直接的室内设计，没有一点花边和粉脂的东西，不像其他舍主那样张扬，或者帖些时尚元素来给苍白增色。“Packages”，“Commands”，“Getting Started”，每个柜子都那样简单有条理，吸引我仔细的翻拣审视摩挲把玩。

2010年11月11日

快呀。爱用爱因斯坦关于时间女人和炉子相对关系的论述来表达我对如梭光阴的理解。半年的好奇不解兴奋困惑醒悟感慨如同第二次恋爱。因为爱过恨过悔过谅过无奈过直到淡淡的过到如今才知道真爱。

他哪里吸引了我？

他哪里吸引了我？

简单直接，富于表达力，强类型，执行快速，以及，他的背包。

不信？先来个刺激的。仔细看下面的程序，大声朗读。电脑语言首先是说给别人听的。

```
package main
import "fmt"

func main(){
	i, o := make(chan int), make(chan int)
	go func(){
		for {
			o<-<-i
		}
	}()
	i <- 0xBabeFace
	fmt.Println(<-o)
}
```

你相信他是C二代吗？不懂才好学。半懂不懂超稳态，不急，早晚会到的，学的再精也会忘一半，所以我们刺激之后慢慢来。

他的背包。

```
package main
import "fmt"
```

一个完整的Go程序是一系列package包们import导入在一起写成的。main 包可以编译为一个可以执行的文件。 “fmt” 包被导入才可以使用其中的Println函数，写为fmt.Println。 读为fmt的Println。那一点儿

代表的可是从属关系。就是说，这个Println函数不是随便哪个包的，也不能不属于一个包，这里的Println是fmt包的Println函数，正如这里的main函数是我们自己的main包里的函数一样。

语义重复一下就是说，Go程序当且仅当由包构成。package在源文件的第一行声明一个包名，在其后import此文件中要用到的其他包名。文件中出现的每个名字不是属于此包，就是属于那一点儿前的包。

他的背包深深的吸引了我的注意。我还记得不论java，C++或C#那类的class，想要找个名字的出处是多么的难。我要先在此函数里找找，再找找此文件，再打开此文件include的头头文件们找它的class一直找到它继承的祖宗八辈的class们。如果没有个完备的IDE，这对我这型惯用简单编辑器者简直是Mission Impossible。

函数，函数。

我真希望他们不叫函数。什么是‘函’，又怎样和‘数’发生的关系？英语的function还可以翻译成‘功能，职务，运行，起作用’，偏偏他叫函数，简直能高深到一下子吓跑一半16岁以下及文科以及数学80分不到的待粉丝们，包括我。这样吧。从今天晚上10点钟开始，只有func，没有函数。不过，我要小心，不要一直读 fukc fukc。

main包的func main(){ 是每一个程序的入口。每个func后是他的名称，再后是他的argument或称parameter的东西都用括号口袋装好。要是你对别人的口袋充满好奇，就像我，很快我们的窥视欲就有的满足。但main的括号口袋里是不能放东西，这是他的签名，Signature。一个func的签名就是他的名字加上口袋里能放什么东西（钱包，手机，车钥匙）还是（口红，镜子，套套），以及他发生功能后能带回什么。

注意到｛尾巴了吗？别的方言里她可以放在一行的开头，甚至自己霸占一整行。Go可对她没那么客气: “你要lady first? Sorry, syntax

error”。有个分号不经意的取代了你的位置。你哭啊委屈啊困惑啊，哪里来的第三者分号？我的程序里没有一个分号的影子啊！

分号啊分号。

她不知道分号是他生活中不可或缺的隐私。每一天的结束，每一行的最后，都藏着那个分号。无所不在，但又隐遁无形。

她问，为什么？他答，不需要知道更好。分号只代表语句的分割，你的身体才是我生命的全部。

{} 笑了。尽管她清楚的知道 func 她的体里的每一行都藏着分号。

i 和 o

```
i, o := make(chan int), make(chan int)
```

我们再大声读一遍：“i和o是make一个chan int和make一个chan int”。再大声读一遍。

发泄完了是不是舒服多了？我来告诉你 i 和 o 到底是什么。喂，猥，不要看着那两个形状的符号发呆了。

int 是英文的 integer 整数的意思。也就是 -10，0，123 这样的数。可不是什么数都是 int，0.1 不是（它叫小数，也叫浮点数），1i 也不是（它叫复数，是 -1 的平方根），2147483648 可能是可能不是（32位的电脑上数太大了所以不是，64位的电脑上是）。

chan是信道，是收发管道的意思。chan int就是收发整数的管道。make是个 func（因为他有括号口袋），他制造make一个口袋类型的值，此处是chan int类型。func make的返回的东西用 := 赋值给此处的 i 和 o。

什么是 i 和 o？我还没有开始讲呢。前面讲的什么int， chan int，make(chan int) 和 io 完全没有关系。耐心，耐心。

我先问你，物理上的电脑是什么？芯片和电路板。我大学里前两年过的尤其浑噩。直到学了汇编语言和自学了Forth语言。我才突然明白电脑无非就是CPU指令和memory内存地址。那些实用的C和Foxpro以及不实用的Pascal和Basic语言等等全都不是写给机器看的。等加入了电子学的知识，我又明白了什么指令和地址全都不是给电脑用的。这个万能的机器只不过是在对一系列有序的电压脉冲刺激作出反应，或点亮显示器的某个点，或让耳机流过电流出个声，或干脆只是改改自己内存的电压。不管它做什么对它都是对的，都是有意义的，只有不同的‘人’对它的行为有不同的解读和期待。

耐心，耐心，就要讲到 io 是什么了。

在我们的程序里，io 只是两个再普通不过的名字。不管你叫她们什么名字，比如 iLoveU 和 oMyGod，她们都只是电脑内存中的两个位置。确切的地方是 func main 运行堆栈中紧接着返回地址的两个地址，存放的是 := 赋值给她们的 func make 运行的结果。赋值的意思就是改变某个地址里的数据（即电压）。

我的意思是说，i，o 是我们起的名字，对电脑的意义只是两个存放1，0这样的数的地址。不多讲了，但看透这层玻璃纸对理解 Go 甚至整个世界的运作都是至关重要的。“道可道非常道”

Ok. Continue.

func 无名氏？go func？

go func(){

我再重复一下上面的主旋律，名字不重要，甚至可以不存在，或者只是个代号 _ 。她的功能代表着她存在的意义。func(){}() 完整的定义且执行了一个无名氏的无意义的生命体： 可以func，无名，空口袋，空身体，执行空口袋。

可我们的无名氏有身体：for { o<- <-i }，她身体的功能是无休无止做一进一出那种事。Go 的循环语句全都叫 for。有四种格式：

```
for {}
for i>0 {}
for i=0;i<1;i++ {}
for i, j := range k {}
```

第一种是此处的永远不死的死循环。

第二种加个结束条件：循环体内可以改变 i 使其大于0 死亡。

第三种除了结束条件，前面加初始值，后面加增量。她实际上是第二种格式的某种比较清楚的写法，等同于：i=0; for i<1 { i++ }。注意分号！这是分号难得的几次登大雅堂的机会。

第四种使用 range 遍历（一个接一个的意思）。

那我们的无名氏不停的做什么呢？我说过是一进一出。我们用分解镜头仔细看看这个动作：

i 是chan int，即一个整数的管道。<-i 从这个管道拿一个整数。

o 也是一个整数的管道， o<- 把一个整数放入这个管道。

也就是说，<- 执行的是进出操作。右进左出。连在一起的 o<-<-i 从i 的管道拿个整数放入o 的管道。看仔细，两个管道，紧挨着，一进一出。

如果没有 go，程序会一直永远不停的执行此循环吗？

不会。如果 i 管道没有数，程序会停下等待一个数的插入。如果没有此处的 go 语句。程序会等待而不是循环。

go 语句把 func 推到后台执行。这样，我们的 func main 得以继续。把 0xBabeFace 插入 i 管道，使后台的 go func 结束等待从 i 管道拿

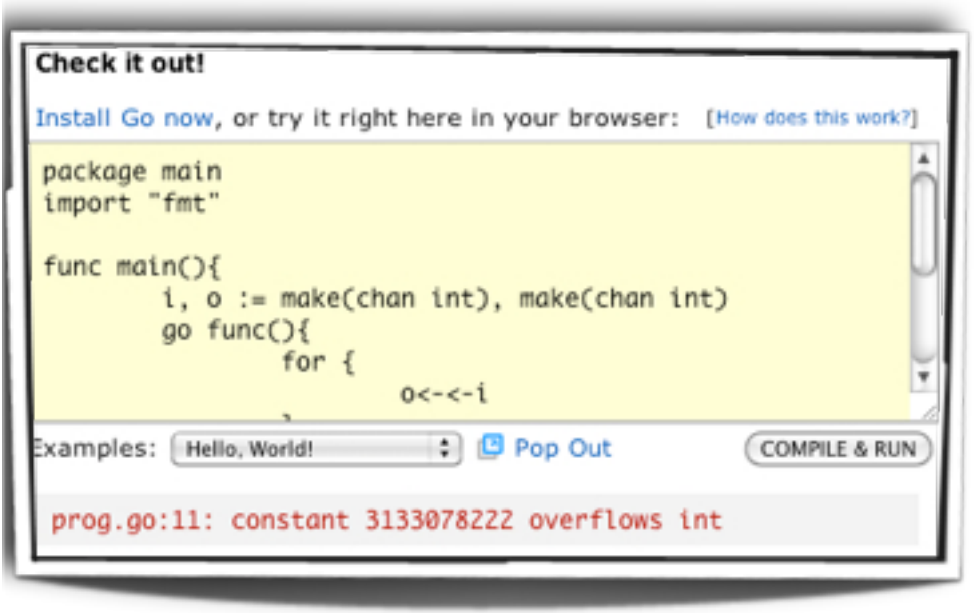
到插入的 BabeFace，放入 o 管道。等待，等待我们的 func main 用 <-o 从 o 管道拿走 BabeFace，进入下一个循环，等待，等待那永远不会来的第二个整数。

最后，我们的 func main 从 o 管道取得一个整数，用fmt包的func Println 打印在显示器上。} 代表 func 定义的结束。func 执行到处返回。

Oh Baby Baby

看完了 AV，你把她抱到 http://golang.org 的游乐场真枪实弹吧。我要继续临窗望海，继续喝我的 Espresso

我好像听到一声狂，不是那种兴奋后的声音，而更像被欺骗玩弄后的狂躁。



我听见她的抱怨 - 你的 0xBabeFace 让我溢出了。

无奈， 0xBabeFace 太大了。换个小小的 0 蛋蛋吧。蓝色的Compiling and Running，蓝色的0。至少你 Go 的第一枪有个结果，尽管只是个正确的零。

第二章

2000年2月21日

Rob 有种被抢的感觉。尽管知道他的每个人也都知道他带着深度近视赢得了80年莫斯科奥运会射箭银牌，可每个人也知道加拿大抵制了那次比赛。尽管Brian和他合著的一本编程实践的书成了公认的经典，但横空出世的Java和C++却要求编程实践作出很大的改变。他也并不很介意和Ken发明的UTF-8编码被冠在IBM头上。他不能接受的事实，是他和Dennis，Ken等设计完成的Unix二世代只能禁锢在实验室里，而草根的Linux俨然完成了Unix的二次革命。

今天要到犹他州发表演讲。Rob准备要呐喊：系统软件研究已不合时宜。原因有，硬件极速发展而软件停滞不前。微软引领了绝大部分的发明而Linux只是炒旧饭。观察比较而不是创新占据着学界。互联网和起步公司吸走了新血，而老祖母也要电脑使整个工业系统要服务最普通的人。系统软件的研究已经无关紧要，我们能做的只有改变。我们要准备好。“Gold rush leaves ghost town”。比如，世界的未来是分布计算，但语言界还没有响应这一需求；互联网占据了信息的发布和使用，但要用户主动获取，而我们可以让信息找上用户；大量的组件讨论只得到Unix管道这一成功案例，我们应有办法用散件搭建成互动的分布应用程序。

Rob有些无奈。他在贝尔实验室潜心设计完美的操作系统，为朗讯研发机顶盒虚拟机及其语言，他都能得心应手。但市场，或说没有市场，使这个顶尖的计算机科学家觉得自己并无用武之地。这并不会影响他的研究，只是他觉得，研究的方向要适应大环境的变化了。

2007年5月2日

helloworld 在unix新闻组上问：Ken在google.com有个电邮，Ken到谷歌去干什么？

Ken和Dennis是Unix和C的发明人，是公认的顶尖骇客。他在94年底跑到莫斯科开米格29战机，然后2000年从贝尔实验室正式退休全职飞行。2007年加入谷歌。为了什么？新闻组上没人知道。

2009年11月11日

Joe在Plan9新闻组上问：“Is this the same Russ Cox we know here?”

Russ是Plan9圈子的名人。也可以说是Ken/Dennis/Rob的Unix衣钵的传承人。读高中时就连续三年代表美国参加国际信息学奥林匹克竞赛。然后在MIT直到2008年博士毕业。他把Plan9操作系统的运行环境几乎完整的移植到Linux，FreeBSD和Mac OS X上。在这个圈子里就知识的广度而言无出其右者。

hiro很快答复了一个短片：Rob和Russ正式发布Go编程语言。

现在一切都清楚了。2007年Ken加入谷歌是为了与Rob和Robert一起研发Go。Ken是Plan9的C编译器的作者，所以很快在半年内完成了Go语言的编译器。Go的语法和运行环境也基本定型。Russ在2008年加入完成大量的基本包的开发。Ian是GCC的活跃分子，他的加入使新生的Go有了第二个编译器。Ken的编译器可以快速运行生成代码，而Ian的编译器可以生成快速运行的代码。更重要的是，两个不同的编译器使Go的语言设计不会依存于具体的实现。语言可以在概念上更规整，同时也更接近编译器下面的机器。

有人说C语言的偶然成功是因为她飞行在合适的高度。既不会离底层太远降低机器的效率，也不会离高层太远降低程序员的效率。C把寄存器的分配交给编译器自动完成，而让程序员紧紧抓住内存地址的分配和使用。电脑的寄存器只有十几个，编译器可以轻易的达到出色程序员的水平，从而使程序员从繁琐的记账式的汇编语言解脱出来，得以用C语言完成数百万行代码的程序，使电脑真正成为全能的计算器。

但很快机器的性能大大超越了程序员的驾驭能力。Bill Gates曾许诺的640千字节内存能满足人们的全部需要，很快的变成了笑话。现在最普通的电脑也有了比640千字节大一千倍的内存需要程序员用C语言紧紧的照顾。这几乎是不可能任务，所以电脑病毒，死机，系统崩溃就像了黑死病一样蔓延。而程序员就像中世纪的牧师一样只能诅咒撒旦再让病人自己孤独痛苦的把剩下的几十年压缩在几天过完。

当然红衣主教们不能熟视无睹。普遍缺乏或有意避免科学，他们泡制了C++， Java，C#等等天价药材。程序员信徒般虔诚的传播福音，世界有救了。当然世界更糟了，人民又被剥削和欺骗了一遍。Y2K那个世界末日让人们疯狂的救赎自己，肥了谁？信息的高度不对称让信息科学就像经济学一样成为打着科学幌子让世界混乱让极少人自肥让所有人买单的一场金字塔传销骗局。

只有革命才能改变当权者。Y2K之后，草根的Linux燎原了，全部的Unix超级变种被烧个精光。草根的Python和Ruby也借风般扶摇直上，大有主流之势。互联网终于彻底颠覆了中世纪的教权。平等，开放，共享，免费食堂，世界哄烈的大跃进了十年。而三年自然灾害如风般淡过，才有人陡然发现，世界更混乱了。

新库大量涌现且高度相互依存。多核的CPU，多机联网的架构使得并发执行成为必须。而十年间却没有新的有效的编程语言。大量宝贵的程序员只是在兢兢业业的维护着某个脆弱的小世界不会突然坍塌，还要推着它快速旋转跟上大世界的变化。

就像黑暗中世纪后文艺复兴带来的昌明，使牛顿和高斯等能奠定现代科学的基石，Go五人组的工作也可能成为后世的宝贵财富。如果信息科学由此得到系统的整理，新人得到充分的启蒙，系统软件的研究将不再无关紧要，电脑将能在新的世界有序高效的飞翔。

刚刚想到这里，我醒了。桌上垂涎一片。

第三章

“妈，每个同学都有iPhone了。我攒了些钱，可还差一千。我想感恩节去买一个。”

照例女儿的电邮最后会很乖巧的讨些钱。照例我会很大方的寄很多给她。远方的女儿啊。小小年纪就寄宿，妈哪里会不宠你。只是，你的信怎么越来越少呢？

iPhone真是好。就像我的女儿。我曾很抗拒女儿的出生打乱了我平静安逸的日子。挣扎了半年后，女儿带给这个家庭的快乐，就如同大吃海鲜一样狼藉而满足。知道女儿离开身边，才知道那曾经不期而至的变得如此不可或缺。

从iPhone开始，我信奉了Apple的产品。当得知Rob和Russ也在用Mac开发Go时，我的信仰更加坚定了。Go在Mac OS X上会得到最优的实现。

女儿不在身边了，才发现日子可以有多么的空。突然想，最空的Go程序是什么样子。没什么可以活在真空里，总得有个基本环境。

```
package main; func main() {}
```

用 6g -S 可以看到汇编代码。我穿插着自己的注解。

```
$ 6g -S empty.go
```

```
--- prog list "main" ---
0000 (<epoch>) TEXT    main+0(SB),$0-0
```

啊main。0000 是指令计数，不是地址。TEXT是说下面的指令放在代码段

```
0001 (<epoch>) RET     ,
```

RET 指令从main返回

```
--- prog list "init" ---
0002 (<epoch>) TEXT    init+0(SB),$0-0
```
啊init。每个包可以有零到多个 func init() 完成包的初始化。
```
0003 (<epoch>) MOVB    initdone·+0(SB),AX
```
initdone存放的值放入寄存器AX。 initdone在下面有声明。
```
0004 (<epoch>) JMP     ,6
```
跳到第六条指令。
```
0005 (<epoch>) JMP     ,18
```
跳到第十八条指令。看看从哪里跳来此处的。
```
0006 (<epoch>) MOVB    AX,BX
```
这里是第六条指令。把AX的值放入BX
```
0007 (<epoch>) CMPB    BX,$0
```
比较BX的值和0
```
0008 (<epoch>) JEQ     ,5
```
如果相等，跳到第五条指令
```
0009 (<epoch>) JMP     ,11
```
否则，跳到第11条指令
```
0010 (<epoch>) JMP     ,16
```
跳到第16条指令。从哪里跳来的?
```
0011 (<epoch>) MOVB    AX,BX
```
这里是11条指令。把AX的值放入BX
```
0012 (<epoch>) CMPB    BX,$2
```
比较BX的值和2
```
0013 (<epoch>) JNE     ,10
```
如果不相等，跳到第10条指令
```
0014 (<epoch>) RET     ,
```
否则从init返回
```
0015 (<epoch>) JMP     ,16
```
这里是15条指令， 跳到第16条指令，问题是，从哪里跳到这里的?
```
0016 (<epoch>) CALL    ,runtime.throwinit+0(SB)
```
这里是16条指令，执行runtime包的func throwinit

```
0017 (<epoch>) JMP     ,18
```
执行完毕，跳到18条，又一条多余的指令
```
0018 (<epoch>) MOVB    $2,initdone·+0(SB)
```
第18条，把2放入initdone地址中
```
0019 (<epoch>) RET     ,
```
第19条，返回
```
0020 (<epoch>) RET     ,
```
第20条，多余的返回。
```
--- prog list "<S>" ---
0021 (<epoch>) GLOBL   initdone·+0(SB),$1
```
GLOBL声明一个地址，给initdone用来放数1
```
0021 (<epoch>) DATA
type.func()+0(SB)/8,$type.*runtime.FuncType+0(SB)
```
下面一行行都放着一些类型的信息，供程序参考，可惜我们没用到
```
0021 (<epoch>) DATA
type.func()+8(SB)/8,$type.func()+16(SB)
0021 (<epoch>) DATA    type.func()+16(SB)/8,$8
0021 (<epoch>) DATA    type.func()+24(SB)/4,$4135763190
0021 (<epoch>) DATA    type.func()+28(SB)/1,$5
0021 (<epoch>) DATA    type.func()+29(SB)/1,$8
0021 (<epoch>) DATA    type.func()+30(SB)/1,$8
0021 (<epoch>) DATA    type.func()+31(SB)/1,$21
0021 (<epoch>) DATA
type.func()+32(SB)/8,$go.string."func()"+0(SB)
0021 (<epoch>) DATA    type.func()+40(SB)/8,$0
0021 (<epoch>) DATA    type.func()+48(SB)/1,$0
0021 (<epoch>) DATA
type.func()+56(SB)/8,$type.func()+88(SB)
0021 (<epoch>) DATA    type.func()+64(SB)/4,$0
0021 (<epoch>) DATA    type.func()+68(SB)/4,$0
```

```
0021 (<epoch>) DATA
type.func()+72(SB)/8,$type.func()+88(SB)
0021 (<epoch>) DATA    type.func()+80(SB)/4,$0
0021 (<epoch>) DATA    type.func()+84(SB)/4,$0
0021 (<epoch>) GLOBL   type.func()+0(SB),10,$88
0021 (<epoch>) DATA
go.importpath.runtime.+0(SB)/8,$string."runtime"+0(SB)
0021 (<epoch>) DATA    go.importpath.runtime.+8(SB)/4,$7
0021 (<epoch>) GLOBL   go.importpath.runtime.+0(SB),10,$16
0021 (<epoch>) DATA    string."func()"+0(SB)/7,$"func()\z"
0021 (<epoch>) GLOBL   string."func()"+0(SB),10,$7
0021 (<epoch>) DATA
go.string."func()"+0(SB)/8,$string."func()"+0(SB)
0021 (<epoch>) DATA    go.string."func()"+8(SB)/4,$6
0021 (<epoch>) GLOBL   go.string."func()"+0(SB),10,$16
0021 (<epoch>) DATA    string."runtime"+0(SB)/8,$"runtime\z"
0021 (<epoch>) GLOBL   string."runtime"+0(SB),10,$8
0021 (<epoch>) END     ,
```

直到结束还全是第21条

果然空既是色。何为色？就是那些看得见摸得到的。并非特指女色情色。我正在色迷迷的看着这个空程序。要把他看破。从此不再生色。

至少我看破了此三点（非彼三点也）

1. 汇编真不是给人写的。
2. 编译器写的汇编真不如人写的。
3. 能写编译器的人真不是盖的。

色心未死，把上次的io又叫来折腾一翻。嘿，无名氏给了个编号叫_func_001。程序增大到106行指令，而后面的类型信息更是大大增加。神也不能手编这样的程序吧。

眯眯的看他都叫了哪些运行包的 func

```
$ grep runtime io.s
0002 (io.go:5) CALL    ,runtime.mal+0(SB)
分配内存给 i
0006 (io.go:5) CALL    ,runtime.mal+0(SB)
分配内存给 o
0011 (io.go:5) CALL    ,runtime.makechan+0(SB)
初始 i 为管道
0016 (io.go:5) CALL    ,runtime.makechan+0(SB)
初始 o 为管道
0028 (io.go:10) CALL    ,runtime.closure+0(SB)
go func 之前最好套套。closure叫闭包。
0034 (io.go:10) CALL    ,runtime.newproc+0(SB)
啊newproc，在后台go func
0041 (io.go:11) CALL    ,runtime.chansend1+0(SB)
管道入
0043 (io.go:12) CALL    ,runtime.mal+0(SB)
0051 (io.go:12) CALL    ,runtime.chanrecv1+0(SB)
管道出
0055 (io.go:12) CALL    ,runtime.convT2E+0(SB)
类型转换给fmt.Println
```

很好玩，长见识。可太伤身。就此色既是空了。被阿罗抱过河也不菲菲想了。

第四章

2010年5月14日

有人在golang-nuts发帖，说“hg log -r1 -v”发现Go的第一夜给了Brian，时间竟然是1974年，也就是说是她上辈子的事情。Russ回帖说实际上她的第0夜发生在1972年，那是她还嫩，刚见世面，只是说个‘你好，世界’。她连C都不行，只是个B。全名BCPL。那时代干活不容易，要打卡，要到处打洞。

我就想，CSI一下看：

```
$ hg log -r0
changeset:   0:f6182e5abf5e
user:        Brian Kernighan <bwk>
date:        Tue Jul 18 19:05:45 1972 -0500
summary:     hello, world
```

72年，UNIX刚两岁，刚用C重写过

```
$ hg log -r1
changeset:   1:b66d0bf8da3e
user:        Brian Kernighan <bwk>
date:        Sun Jan 20 01:02:03 1974 -0400
summary:     convert to C
```

不是编的吧，哪会1点2分3秒那么准?

```
$ $ hg log -r3
changeset:   3:172d32922e72
user:        Brian Kernighan <bwk@research.att.com>
date:        Fri Apr 01 02:03:04 1988 -0500
summary:     last-minute fix: convert to ANSI C
```

更不靠谱，挺了14年才last minute。时间也是那么精准。

```
$ hg log -r4
changeset:   4:4e9a5b095532
user:        Robert Griesemer <gri@golang.org>
date:        Sun Mar 02 20:47:34 2008 -0800
summary:     Go spec starting point.
```

这个我信了。零八年西八区星期天晚上八点后。走规范开始in g点。

```
$ hg log -r5
changeset:   5:f7ab39d05bcd
user:        Robert Griesemer <gri@golang.org>
date:        Sun Mar 02 23:02:36 2008 -0800
summary:     - added language for map and channel types
```

半夜就找到了地图和地道

到现在的一千天里，这种进行性行为已达6千次之多，平均每天6次（不包括最初3次），会员也近4千。数字的背后，是受红的程度。2009年年度最佳金鸡新羽奖（TIOBE）。2010年最开放百花奖（BOSSIE）。如日中天，欲满全球。

2010年10月18日

Rob春风得意的在10年度Java搞对象大会（JAOO）上发表演讲。他不知道的是，会后，JAOO不带JAVA和OO玩了。新名很时髦，加进了GO元素，就简单的叫GOTO。

Rob说：Go受欢迎是个愉悦的意外。

他是有惊喜的意思。被抢去的钱夹神奇般的又捡了回来。

Rob补充到：但事后想想，我们接近Go的设计手腕是使Go善解人意又多产的关键。

他进一步阐明，Go是讲原则的。Go的格言是:

1. 简单。想法一说就明白（但行动仍会细致入微）
2. 正交。多个想法可以干净混用，容易理解和预想进行的动作
3. 简短。少废话，不用干什么都先讲清楚
4. 安全。粗鲁的行为能很快觉察

这些综合起来使Go能善解人意。

Rob强调：尊重！

Go相信程序员讲的和想的一样。作为回报，Go会试着遵从程序员的意图。安全有趣的玩法是可行的。这是安全带套和防轮间的区别！

Rob用数字说明，钥匙词的数量大约可以衡量麻烦程度。他给的数字中，最麻烦的是一个叫操夹夹圈儿插的家伙，要用72个钥匙词还要多付11个给预订词及另类拼法。那个叫操筛的也不是省油灯，多达72个钥匙。而我们可爱的Go只需25把钥匙就完全开放了。

紧接着带我们参观了Go的网舍:

```
package main
import (
	"fmt"
	"http"
)
func handler(c *http.ResponseWriter, r *http.Request) {
	fmt.Fprintf(c, "Hello, %s.", r.URL.Path[1:])
}
func main() {
	http.ListenAndServe(":8080",http.HandlerFunc(handler))
}
```

他说，这是“你好，世界” 2.0 版。在本地主机8080端口服务，意思是说，网舍地址是“http://localhost:8080/world”。

Rob没有细讲Go在那个网舍里做什么。我觉得挺可惜，就武断的补充一下：和之前io的大同小异，只是用了http包的func ListenAndServe。这个func会瞪着耳朵听谁来找8080号，然后把他带到http包func HanderFunc准备的一个handler，再回来继续瞪耳朵。那个handler动手的有两个口袋，一个要装来人的c（钱），另一个接受来人的请求。动手的会用fmt的Fprintf，放钱进去，再把请求的URL的第一部分（也就是world），加上自己该说的“Hello ”，也放进去。这样，是fmt的Fprintf在来人的浏览器上显示“Hello， world”。相比较72年初夜的那个Hello，她成熟了很多，有能力绕地球几圈动用上千台机器说那句经典台词了。

Rob越讲越兴奋，加了几个段子，可惜我没在场，录音又非常可疑的出了非系统软件的故障。我可从他的稿子里什么都猜不到。

突然兴致乏乏。后面大段大段的术语，被我昏昏入睡了过去。

第五章

趴奋。条件反射的惊醒。不需用想，粉红兔子远程投放的粉笔头准确的敲打了一下我的桌面。只好换个体位竖起双眼空盯着黑板继续游梦。

粉红兔子是我们的政经教授。爱用粉笔对我们旁敲侧击，且总是眼带血丝而得名。奇怪为什么今天是白板写满了花花绿绿的代码？

粉红兔子说，今天我们要深入浅出的讲解Go最出色儿的一个设计：Intercourse xxx xxx face。对不起，对不起说走嘴了。是Interface，界面的意思。大家知道Java之类要靠，类的继承，甚至多类的继承，来表达类似却不同东西自己的关系。也就是说，很多时候最麻烦的时候就是在扩张和维持这张关系网。Go也讲关系，但不是什么换对象，而仅仅是协议解决。深入之前的前戏我们来看张图。

```
package main
import (
	"os"
	"io"
	"bytes"
	"encoding/base64"
	"compress/zlib"
)

var tryme =
`eJxslTGy4zAMQ/ucIr00PoDUuWPhRoUkHoE97z8LULI3+f9zJj92dvwMghD
3dV53nazrbGcbc4zzaqKzHlH5OJJ3sSo1v3Pvs+to/5+8K2D7E7DrnH3OcY4
pqYpWcIJWrKubFNx00ISv+wV7VMXXdQ0VNxNUKrmY8mnAck4+3Zy3Apr0v5T
dsI0capZqMpmaSinV03G836TVIt0Nt7kSpm09891m9Bhusbonsqrpm FMFNm1
WqZYEqvMBvR0tjz9gyzEC8aqzKXRVrxWiUrLklgkDqpprHzABJhZc9vm7Tyi
j6RQXMChDa2AVOJbM8pusyt5RSTBnt1Lc+/w9z9fZQhU/Dbgh6KZm2gUtafV
ItGEerOqK7sWRG8z8T89IXI1OceVjJTlGB1SMMptHSHhJAxwxgVCP5748W/m
6Liq7WtfOLuE3WlyZBcM0hTDC+I9W4439SxtCC6co6WYi/jClQsjDYqfw3BE
WGEc3V8tIzg8YK7xnOqLZhmmmxcITQBEKt2hVJWSrxYBkftJeULXND3nhHQ9
SJD9DAWFPu7hcN2Gdi/TPuBG2Z0B1y9DJ0CJm9KWAtgEEbdhCMbvAPRF50X8
GI7rdvw9iENkI15L2UQvFDKsV2FF9fsMoip1G01fD6O5MxMWHpm0Wc+alJKT
vSI8yOh/5D9RSBonO9j7qE7VkzVSgGoOVH8cp2iRrwbDRYMdS911MGXcKTuZ
Ur8WKjud0s819kO6lhrWoOJ08NpGDr0LOiMLBFHOc+zW1e6DcZytne6ADS8u
whjBIBAtQHsz4w8CGWYpjiU1lZrhobTxnPpbjksa/QC0Wk47pY1diva2Ngcc
dyWK2FL/wVrG8sWiE38DFQV+yGmxQwfv4UjyK4wzv1lXocBSGOBoFoxz/EcA
7bBDBz4S9/gUAAP//yf/Baw==`

func main(){
	r0 := bytes.NewBufferString(tryme)
	r1 := base64.NewDecoder(base64.StdEncoding, r0)
	r2, _ := zlib.NewReader(r1)
```

```
        io.Copy(os.Stdout, r2)
        r2.Close()
}
```

tryme是base64编码zlib压缩的一张美图。基于职业操守我不能直接印在这里，你们可以到golang.org的操场上自己跑跑看。你们实习一下吧。

好了时间到我们继续。从大部分的同学的表情可以看出你们还是接受这种教学方法的。现在我们要深入了。注意。

Go的界面说白了就是个功能清单。比如，不管是谁，只要能读会写，能满足这两个功能，我也不管她长的什么样，或还有什么其他功能， 只要她能读会写，就可以在这里工作。

我们倒着看main。

```
        io.Copy(os.Stdout, r2)
```

io.Copy的工作是抄写。她需要一个Writer和一个Reader，这样她用Reader.Read读再用Writer.Write写。此处的Reader是r2，Writer是os包的标准输出，对应的显示器。意思是把r2读到的东西显示出来。r2 Close的时候，你们就看到那个了。

```
        r2, _ := zlib.NewReader(r1)
```

r2 是zlib包的Reader。她从r1读然后解压缩。那个下划线的地方本来是放出错信息的，此时我们不担心会出错，所以用下划线表示。

```
        r1 := base64.NewDecoder(base64.StdEncoding, r0)
```

r1是base64包的Reader，尽管她是用NewDecoder得到的，只要她有个func Read，她就可以做Reader。她从r0读，用Std标准展开读到的base64编码。

```
r0 := bytes.NewBufferString(tryme)
```

最后是r0。她把我们base64编码zlib压缩的tryme字串放到一个Buffer，是因为字串没有func Read，而Buffer有。意思是字串tryme不是Reader而Buffer之后就是了。

这样，Copy要读的时候，用r2.Read，　而r2.Read会用r1.Read，然后r1.Read用r0.Read才最后读到些东西，解码解压然后写在我们的显示器上。

还不明白的同学请起立。很好，不要勉强。我会保障你们最低及格分数的。大家活的都不易。你们可以出去了。剩下的同学把你们的结果图抄下来，自己试着把她编码压缩回tryme。

一声惊雷，我睁开双眼无神的看看玻璃壳外。雨还在刷着漆黑。

第六章

2010年10月5日

昨天是我的生日吗？日历上是，可又有什么意义呢？生日是给别人的。父母生了你，一辈子都会记得那一天的。男朋友记得是因为要头痛去哪里吃晚餐。自己有时还记得是因为生日过去了还没有人记得起。

蜗居的人只能和自己喝咖啡。再投胎转世吧，或能成为个被欢迎型。

Type和cast，就是Go的胎和转世。电脑的1和0就像精卵，靠自然界最强的力结合在一起，两仪生四象生八卦直至无极。电脑看到的也是无穷尽的1呀0呀的bit位的序列。Type，英文本义是字模，就是活字印刷的字，把8个01的bit划定为一个byte字节，进而把这8位的字节分为两类，int8和uint8。uint8叫unsigned无符号型8位整数，可以表示0到255之间的一个数。int8叫signed有符号型8位整数，可以表示-128到127之间的一个数。同样的10000001，如果为int8，它是-127，如果投胎为uint8，它是129。对于有符号数，最高位的0或1是它的正负号。所以int8的10000001是个负数，而后面的7位0000001要加上-128才是它代表的数-127。这种人为的安排叫2的补码，使得电脑做加减时不用区分int8和uint8。10000001 + 01111111 = 0，如果是int8，是-127 + 127 = 0，如果是uint8，是129 + 127 = 256 = 0因为8位最多只能装255即11111111，再加1就溢出，效果就象射了一样。

但这样安排只适用于加减。拿int8和uint8去比较大小是不和谐的。因为它们无法代表广大的从-128到255的整个阶层的需要。此时，程序员这个神要武断了，或着把10000001投胎cast成负数-127使它小于01111111这样的正数127，或者cast为正数129，使它大于127。无论怎样选择，都有选择错误的可能，但这是神旨，远好过电脑自己的选择。

这就是Go坚持的信念之一：凡是涉及不同胎型type的，程序神人要明确决定投哪个胎。胎型不对的不可乱交，想人狗交者，死。但变人为狗交者，性莫大焉。

同理，16位的有int16和uint16，32位有int32和uint32。64位的有int64和uint64。现代电脑科技还不能有效处理64位以上的胎型，所以Go的整数型只有这8种。

小数，也称浮点数，有32位的float32和64位的float64两胎。他们能代表天大范围的数字，例如，给地球的每粒沙子一个编号，是不可能的，但表示最细的沙子和最重天体的质量却可以使用32位的float。

还有64位和128位的complex复数胎型，据说是ken神坚持下的产物特适合某种操作。

Go的全部数值胎型就这些，为了方便时方便，uint8也可写为byte。uint，int，float，complex可以代表32位或64位，因为程序神人不想劳神去决定123这样小小的数到底是int32还是int64，所以就说随便你编译器int儿去吧 ‘cause you are so good.

还有个uintptr的怪胎我现在还不想碰。

为了进一步提高服务水平，我们的Go还能提供自动添一的iota操作。看个例子，

```
const (
 周日 = iota
 周一
 周二
)
```

我有说过Go只懂英文吗？那怎么在国际话大都市混。比如来个阿拉伯，噜咕里叽的一通你就不接活了吗？Go懂世界鸟语，她懂有人从

右向左，有人从上到下，还有人转着圈的写。她说来来来，老娘天生UTF8，你们都等我趴下。

iota过后， 周日一二是0，1，2了。具体什么胎型？等有了assignment行为再说。还没插呢怎么决定DNA？

所以，Go的const常数是广义上的无常数。意思是程序神只管随便开支票，编译器安排投胎才能兑现或跳票。例如1e9，一亿九不嫌多，可别把它当浮点，seconds := time.Nanoseconds()/1e9 它是int64，把纳秒换秒的兑换率。

刚提到Go天生UTF8，她就凑过来“Hello世界”。她说那个string字串，看穿了就是5个byte的Hello和6个byte的世界。看你不懂的样子她幽幽地说，谁叫English强势先霸占了我。早生一千年你们都得学着罗马数字作算数，写标准隶书的程序，对吗 true or false？这可是个问题，需要用到boolean布尔胎。

没了。就这些基本胎型。剩下没提的还有几个复合胎。太累了，先睡一下。

第七章

Rob说Go只需25把钥匙解开全身的锁，我们看看知道了那些:

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

熟脸的有func，interface，go，chan，package，const，range，for，import，return和var。11个。其中package和import用于包的管理。const，var和chan是常量变量和管道。go func return是关于函数func的意思。最后for和range是程序流程控制的一种，是用来控制程序的重复执行，循环的意思。

电脑的爸爸是冯挪伊曼。精神教父是病人图灵。祖宗可能叫算筹，但算筹独立演化为代表先进力的算盘。两千年前算盘统治东北半球时，电脑连个想法都还没有呢。

当年我们电脑的冯爸爸一穷二白也在搞氢弹时，世界第一胎电脑爱妮哑客在他的参与下诞生了。冯不是她的唯一爸爸，但他写了篇报道:爱妮报告草稿第一版。没想到啊没想到，就凭这101张纸，经大肆炒作后确定了冯爸的地位。其实，纸上只是记录了制造爱妮时边干边啄磨出的一些道道，例如，他指出，数和码要一起放在电脑子的同个楼盘，即所谓的stored-program囤控的概念。当然，冯爸的出位手段遭到爱妮爸爸组的痛批：其一，先期未经组织同意使幼齿曝光，直接导致patent专属权的丧失。其二，正如一些组员爆料，囤控的概念是冯爸在宾州大学电工学校的几个会上听来的，然后用他所擅长的官方逻辑语言加加工，就可以在报告上省掉其他人的名字了。

比起美国人冯，英国病人阿兰图灵要圣洁的多了。虽然他没有掺乎爱妮被造事件，精神上他很早就给出了基本原则。他说的很精辟:

“在无尽的纸条上画方格可得无尽容量的记忆。每个方格打上个记号。任何时刻机器只给一个记号，叫作扫过标记。机器可以更改扫过标记，其行为是此标记决定的，而纸条上其它地方的标记不会影响到此刻机器的行为。但是，纸条可以被机器前后移动，这将是机器的原始行为之一。因此，纸条的每个记号最终都会被轮一次。”

太出彩了。图灵灵的结果只能病了。当时打希疯子打到人人都疯了，所以被疯子说疯真是不稀奇。图灵帮着破了希疯的密码，又在国家物理实验室设计了最初的几个囤控电脑之一，但42岁的他，最终灵到被服了毒。毁掉肉还不够，40岁的他被指过分喜欢同性，触犯了国家大法，因而被施以女刑。太惨了，英版袁崇焕。可是英版乾隆的出现要等到死后55年，而且是迫于互联网行动的力量！

不难理解整个电脑界是多么的震惊和痛惜。图灵死后不久，皇家协会授予他大英帝国最高勋章。美国ACM计算机器协会的电脑诺贝尔奖从此以他为名。获得图灵奖成了电脑科学家的最高荣誉。

图灵贡献的就是Go的if，如果的意思，和对应的else，否则的意思。它们用来做可控的前后移动。例如:

```
package main
import "fmt"
const (
	聪明人 = iota
	英国人
)
var (
	图灵 = 0x900d
)
func main(){
```

```
	if 图灵 == 聪明人 {
		if 图灵 == 英国人 {
			fmt.Println("和谐")
			return
		}
	}
	fmt.Println("图灵可以不和谐")
}
```

由于世界语的关系，所见并非所得。Go天生UTF8，所以保存文件时必须使用UTF8编码。不可用Unicode或国标等。

另外，Go强制使用{}，并且 { 必须在行尾。道理和func是相同的，野鸡分号的原因。

==是相等的意思。如果我早点告诉你 && 代表并且，|| 代表或者，!= 代表不相等，func main可以写成:

```
func main(){
	if 图灵 != 聪明人 || 图灵 != 英国人 {
		fmt.Println("图灵可以不和谐")
	} else {
		fmt.Println("和谐")
	}
}
```

改改意思，还可以写成:

```
func main(){
	switch 图灵 {
	default:
		fmt.Println("图灵可以不和谐")
	case 聪明人:
```

```
            fallthrough
        case 英国人:
            fmt.Println("和谐")
        }
}
```

这样我们又一下子打开了四把锁：switch，case，default，fallthrough，代表’切换‘，‘分支’，‘默认’，‘坠入’。switch根据后面的值从其下分支里选一个，如果没有合适的，则使用default。而fallthrough则继续到下面的一个分支。

这样，只剩break，continue，select，defer，goto，map，struct和type八支钥匙了。break结束for的循环，而continue跳过剩下的部分继续循环，goto直接跳到某处，它们总是和if，switch这样的条件控制在一起用。select用于选择一个活跃的chan。defer是func返回时用的，它们属于高等技能，等老鸟了再说。而map，struct和type，作胎专用。也别只看我一人儿上下翻飞，兴致勃勃，大伙儿一块儿上啊，活学活用，边干边学嘛。

第八章

你说好想回来听我的小故事，可也想去看看落基山。你去吧女儿。20个小时的飞机对谁都是禁闭。你说最近总是不能再无忧无虑。说常困惑自己会怎样怎样。我说女儿啊时间是河秒就是沙随它去吧。记得你爱堆沙堡，看着浪把它们一个个卷走，然后你只是捧起沙，它们就那样随意流下，也能堆成个弯月般的小沙丘。你总是能很开心，不去在意是什么力那么轻易的夷平了你费力构筑的城堡。你好像天生就知道，不管将来会怎样怎样，随心所欲，随遇而安的道理。

落基山几点了？时间是MST还是PST？这老美的时区对华人来说真是个joke。看俺们多简单，无论你在哈密太原还是昆明泰安，青岛银川还是包头武汉，全都是北京时间。我们想当然的事情在美国可大不同。匹斯堡市的火车站有6个时间，每个铁路公司的火车时刻表都用自己的标准时间！当然这是100多年前，但至今美国也没能统一，仍在使用着4个时间。当跨越那3条时间变更线时，可以想象人们是多么混乱。要是你刚好站在那条线上，你可以同时拥有两个正确的本地时间。当然，这仍是一大进步，也是电报和铁路带来的时间需要。在之前，美国使用着8000多个不同的本地时间。

交通旅行还可以用小时来安排时间。到了电脑联网的时代，整个地球表面的机器时间都统一在UTC，也就是一个基于国际原子钟的时间标准。这个钟可不是一个，而是全球70个国家实验室的200多台原子钟在人造卫星的协同下精确对表得到的。

我知道最不可思议的地图时区程序是1992年ioccc国际乱C大奖得主westley.c:

```
$ cat westley.c
```

```
           main(l
       ,a,n,d)char**a;{
  for(d=atoi(a[1])/10*80-
 atoi(a[2])/5-596;n="@NKA\
CLCCGZAAQBEAADAFaISADJABBA^\
SNLGAQABDAXIMBAACTBATAHDBAN\
ZcEMMCCCCAAhEIJFAEAAABAfHJE\
TBdFLDAANEfDNBPHdBcBBBEA_AL\
 H E L L O,    W O R L D! "
   [l++-3];)for(;n-->64;)
      putchar(!d+++33^
           l&1);}
```

这是个完整的C程序，运行时给出纬度和经度则标出其位置，如北京是北纬40度东经116度，显示的结果如下。注意我用红笔标出的引号是北京的位置。

```
$./a.out 40 116| fold -w 80

             !!!!!!!!!!! !!!            !!!   !!!!!!!
! !!!!!!!!!!!!!!!!!  !!!!! !    !      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 !!!!!!!!!!!!!!!!!!! !!!!          !  !! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
         !!!!!!!!!!!!!!            !!!!!!! !!!!!!!!!!!!!!!"! !!    !
            !!!!!!!!!             !! !   !!!!!!!!!!!!!!!!!!!  !
    !        !!!!  !              !!!!!!!!!!!!!!!!!!!!!!!!!!
               !!!!!             !!!!!!!!!!!!!   !!!   !!! !
                    !!!!!         !!!!!!!!!!       !     ! !  !
                    !!!!!!!!          !!!!!                    !!
                     !!!!!!            !!!! !              !!!!!
                      !!!!              !!                !!!!!!!!
                      !!                                   !!  !!     !
                      !
```

好在Go不许我们这样写程序。Go的原则是程序首先是给人看的，其次才是给机器用的。所以Go的写作风格不是随意的，例如，{必须在行尾。甚至有个gofmt的命令自动帮你排版。Rob建议所有的Go程序都用gofmt以达至风格的统一。

我们需要一些原始的C知识才能知道westley.c在干什么。由于Go是C二代，知道C和C的问题才能比较出Go的基因为什么要改变。

main也是每个C程序的入口。后面括号里的l,a,n,d是main用到的变量，即Go的var。其中a的胎型在其后给出是char**,也就是Go的[]string的意思。l,n,d未明确胎型所以默认为整数型也就是Go的int。

C的函数体也是放在{}里，但C没有行的概念，所以每个语句都要分号结束。for也是C的循环语句，但C还有while和do while两种循环，在Go里都是for了。并且C的循环控制要放在括号里，Go的不用。

C没有包的概念，atoi函数在标准库里，所以不需import，而且C的叫#include，注意开始的#号，代表的是预处理的意思。C可以有很多这样的预处理，例如#define，#if，#else，#ifdef等，Go一概取消。

a[1]和a[2]是运行此程序时的命令行参数，即纬度和经度。是两个字串，所以由atoi转换为整数计算得到d的值。另外，C的数组也是从0开始，a[0]是程序名，即a.out。但和Go不同，C数组的下标可以越界，这也是电脑病毒产出的根源。因为病毒也是程序，它会偷偷把自己放在某个数组的后边，再改掉其进攻程序下标值使其执行自己，从而把病毒当作自己的一部分不加区分。更常见的是，数组没有越界保护，程序的漏洞，也就是程序员没有发现的逻辑错误，不能被及时察觉，导致危险的结果。这些危机在Go里都不存在，所以Rob说Go是安全的语言。

和Go一样，for的控制有三部分由分号隔开，初始；循环；增量。此处初始是d=纬度/10*80-经度/5-开始位置。除10是因为每十度一行，乘80是每行80个字符。因为没有加换行符，需要在80列的终端显示（例如用vi），或像我这样，用‘|fold -w 80’处理。

l 是main的第一个参数，其值是2，也就是程序命令行参数的个数。l++使其每次循环加一，l++-3使其从0开始。Go也可以用++和--来自动加一减一，但只能单独使用，不能像C那样和其它运算混在一起导致程序员搞不清顺序犯错误。

```
"@NKACLCCGZAAQBEAADAFaISADJABBA^SNLGAQABDAXIMBAACTBATAHDBANZ
cEMMCCCCAAhEIJFAEAAABAfHJETBdFLDAANEfDNBPHdBcBBBEA_AL H E L
L O,    W O R L D! "
```

这是个无名的字串，[l++-3]下标取到的字节值给n。它代表的是世界地图！循环从第一个字符到最后能自动结束，是因为和Go的字串不同，C的字串尾隐含了一个值为0的字节，可以方便的知道字串结束在哪里，但同时要求字串中间不可以有值为0的字节。当n为0时，for循环结束。

每次for循环执行另一个for循环`for(;n-->64;)`。内层的循环控制显示多少个字符。这需要知道每个字符所代表的数是什么。C使用ASCII，所以@是64，N是78，K是75等等。和Go不同，C隐含的把字符转换为整数。Go必须明确的用int()转换，麻烦了一些，但避免了程序员犯糊涂的错误。明确转换也是Go的一个原则。

putchar是C的另一个标准库函数，用来显示字符，此处为空白，叹号或引号，其值分别是32，33和34。所以有!d+++33^1&1。要知道它在干什么需要知道C的运算优先级，太过复杂。我们只要知道结果就好了。Go的运算优先级得到了简化。

把这个程序用Go写一遍应该不难了吧。

第九章

裆说，自由只有一种，就是君权神权裆权金权严刑峻法之下的平等自由，其从来就只限定于无权的阶层。自由的存在就是肯定不自由，就像生命的意义在于存在死亡。光明的意义在于存在黑暗。那种随时剥夺你存在的特权赋予人人黑暗统治下的自由生命。

我说，自由还有一种，就是每个独立个体在相同的外部环境中物竞天择。羚羊不是天生喂豺狼的，自然淘汰是整个种群健康发展的关键，而豺狼只是在设计中被羚羊使用着。公平是自由的意义。公正是自由的保障。权贵是自由战士存在的意义。严刑峻法只是在强化着自由抗争的力量。

就像台湾，裆说你从了我吧，花儿任你采，果儿任你摘。穿金带银坐地收租不好过你胼手砥足？

湾儿说，你祖上家大业大，谁个不想从？但你老子不学好学人抽大烟，家破人亡把我也卖给个狗日的窑子。你和你哥打架闹分家。你哥被狗日的踹一脚被你打的连滚带爬差点儿断气儿，缩在我这儿还占了我。不是我不想家，是你哥认了个老大。你有本事把他摆平，别整天拿个菜刀晃我，还到处扒我的衣裳。好看哪，世界面前丢自己的脸。

裆说，现在哥快完了，你那儿无赖闹翻天了，他老大也快罩不住了。我再吓唬吓唬他们就放了你啦。

湾儿说，你农民呐有几个钱也还只是个富农。你那钱挣的容易啊让几个败家子儿随便花？看看他们把钱都给谁了？还说吓唬人家老大，他多喷点儿口水就能把你的钱全泡汤啦。你那点儿肌肉哪儿打得过人家浑身铠甲。你先别总想着打架总想着统一我啦，先收拾那败家子儿吧。他们把你都快掏空啦。

裆说，我知道啊但没办法。我已纵欲过度爱滋缠身又癌症晚期了。脑细胞看着还好但下面的全烂啦。每个器官都在疯狂生长挤压蚕食着还在提供营养的身体。偶尔有个清醒的细胞说句话或哪怕只是有个想法，也会被已经过于亢奋的免疫系统残杀。我没救啦，你怜介我赶紧从了吧。

湾儿说礼丧求诸于野体制坏了生老病死就随他去吧。走的只是思想留下的躯体是可以自然分解再利用的啦。生生不息指的从来不是个体。你的土地你的人民是你的未来你转世的希望。不要再蹂躏他们了，毁了他们毛将焉附？你早点儿安心的去吧。维持一个病体的代价会毁掉整个的家。

我说，对于存在于物理世界之外运行的电脑软件，就像人的思维一样，它的意义是编排好的，其本身是无从知晓其意义的，可以随时被终止，可以运行时出错，或本身就带着明显的错误。单细胞的程序对同样的刺激做出相同的反应。再多的个体也会毁在环境轻微的改变。大量细胞的聚合增大了系统容错的能力，但单性繁殖使对环境变化的适应仍旧缓慢。两性的结合，使相隔很远的不同DNA快速融合，加上少量的变异和物竞天择，是现阶段最高级的生命适应环境的方式。这种方式已经自然健康和谐的存在了几亿年，直到类人猿最后一次的突变，导致地球生命的不可逆转的最终自我毁灭。因为，人得到了特权，生命不再自由，特权只会在乎自身短暂的享乐，从不关心他人的死活。代价，他们轻蔑自由无视自然的代价，只有随时的GAME OVER。

第十章

回到电脑，Erlang的作者Armstrong在其博士论文中证明，单个机器自身，由于对其硬件出错的不可挽回，本质上是不可靠的。但独立机器构成的网络，可以互相监督和备份，在容错系统软件的支持下，完全有能力大大提高整体的可靠性到九个9，也就是理论上几乎完全可靠。

Erlang是个出色的并发编程语言。Golang在设计上也是很出色的并发编程语言。假以时日，Go一定可以甚至超过Erlang的完善程度。因为，其一，Go是编译执行，比靠虚拟机操作的Erlang要快很多。其二，Go的语法更接近主流的C，远比Prolog逻辑语言出身的Erlang有更多的程序员。

并发和容错的是一个问题的两面。多机并发保证可靠的出错处理，而可靠的出错处理机制是成功并发的关键。

在Go语言里，go func 是并发的单元。chan 是协调并发单元的机制。panic和recover是出错处理的机制。而defer是神来之笔，大大简化了出错的管理。

先看看go func。她本来也是个普通的可以func。例如一个最简单的例子:

```
package main
func f(){}
func main(){
      defer f()
      go f()
      f()
}
```

```
$ 6l -S go.go

--- prog list "f" ---
0000 (go.go:2) TEXT    f+0(SB),$0-0
0001 (go.go:2) RET     ,

--- prog list "main" ---
0002 (go.go:3) TEXT    main+0(SB),$0-0
0003 (go.go:3) JMP     ,5
0004 (go.go:3) JMP     ,18
0005 (go.go:4) PUSHQ   $f+0(SB),
0006 (go.go:4) PUSHQ   $0,
0007 (go.go:4) CALL    ,runtime.deferproc+0(SB)
0008 (go.go:4) POPQ    ,CX
0009 (go.go:4) POPQ    ,CX
0010 (go.go:4) TESTQ   AX,AX
0011 (go.go:4) JNE     ,4
0012 (go.go:5) PUSHQ   $f+0(SB),
0013 (go.go:5) PUSHQ   $0,
0014 (go.go:5) CALL    ,runtime.newproc+0(SB)
0015 (go.go:5) POPQ    ,CX
0016 (go.go:5) POPQ    ,CX
0017 (go.go:6) CALL    ,f+0(SB)
0018 (<epoch>) CALL    ,runtime.deferreturn+0(SB)
0019 (<epoch>) RET     ,
```

f()很普通，可以被直接func，也可被go func，还可以被defer func。从汇编码我们清楚的看到，第17行一个简单的CALL呼叫可以就地func。而第12行到第16行的go func则有些学问。

编译器安排好了一些前戏，把被func的地址，就是$f，和其口袋要放的钱数，PUSHQ存好，然后叫来runtime.newproc这个家伙。他拿着地址和钱给f()一个新房间，让你们不受干扰的好好func。完后，编译器再安排两个POPQ打扫干净。由于有了go func，除了在 main 大厅外，Go还可以轻易的安排非常多的小房间，和main大厅里的同时func。这种互不打扰的群体func行为史称并发。

再看看defer。类似go func，第5和第6行放好地址和钱，第7行叫来了runtime.deferproc。runtime就是龟婆，提供func执行的环境。defer是延迟的意思好似伟哥，她把本来要马上发生的f()行为推迟到最后发生。也就是RET前的第18行。deferproc过后，除了例常的POPQ清洗，还有第10行的TESTQ感染检查。如果AX是0（第一次还是干净的），则继续，直到最后执行runtime.deferreturn。这个家伙可不简单，他和deferproc还有go都是Ken神的杰作。deferproc记下的地址和钱被deferreturn使用，f()被叫来defer func。但和go func不同，不需给他们分配新房间。因为，正在执行的 main func 快要结束了，只需简单的整理一下床褥就可以让f()继续使用了。这就是deferreturn的工作。具体的可以看看Russ博客。

defer有什么用？最常见的是收尾的工作，如，打开的文件要关闭，分配的资源要释放，堰塞的精液要发泄，这些都是defer常日的工作。但她和recover的配合更是如双飞般曼妙。

panic和recover是Rob的杰作。和go与defer不同，她们没在Go的25把钥匙里所以她们不需要由编译器安排工作。她们和make一样是runtime的内建函数。同样的还有append和copy用来添加和拷贝切片，len和cap用来得到切片和字串的长度以及切片的容量。new和make一样分配和制作一个新类型的变量但new返回其地址而make返回变量本身。还有Ken的cmplx，real，imag用来组装和分解复数值。

panic就像二奶一样见光死。我拍你给组织看。你说笑话谁还怕组织，裆菱早准备好了我的recover，还有个defer等着收拾你这类不知死活的举报者。例如:

```
package main
import "fmt"

func main(){
	p()
	fmt.Println("安全回家")
}
func p(){
	defer func() {
		if r:= recover(); r != nil {
			fmt.Println("组织照顾，票数为", r)
		}
	}()

	fmt.Println("叫基")
	j(0)
	fmt.Println("不可能出事儿")
}

func j(i int){
	if i > 6 {
		fmt.Println("太累了，趴下吧")
		panic(fmt.Sprintf("%v", i))
	}
	defer fmt.Println("清理基数", i)
	fmt.Println("已用基数", i)
	j(i+1)
}
```

先想想整个社会是怎么啦，再去golang的操场验证一下吧。

第十一章

这个世界就是个大垃圾场。我们都只是只蚂蚁，蝗虫的大腿蝴蝶的翅膀蜻蜓的眼睛，也只能像个农民一样想想邻居女儿听听收音机。也只能做着黑色的梦穿着黑色的衣。我要扒光你的衣裳，尽管明天你会死在这床上。你告诉我床单很白你觉得这个城市很脏。我明天想要离开，尽管我觉得你还有那么点儿思想。我明天想要离开，尽管你会死在这床上。离开，离开。趁你还年青我们老啦请爱惜你的童贞吧。

张楚啊，你在哪里？

离开，离开。我知道我要去哪里。我知道我要去那里。那里还有鲜花和牛屎。那里还有我存在的意义。

~ 完 ~

终于2010年11月16日